SRI KRISHNA COLLEGE OF ENGINEERING AND TECHNOLOGY

Cognizant Hackathon 2025

BATCH – 1

USE CASE NO – 3

USE CASE - PREDICTING MEDICAL EQUIPMENT FAILURE

OUR MENTORS

COGNIZANT MENTOR - Arul Prakash | 9894812475 | ArulPrakash.R2@cognizant.com | Cognizant

COLLEGE MENTOR - Mr. G. S. Pugalendhi | 9842465747 | pugalendhigs@skcet.ac.in | SKCET

OUR TEAM

NAME	COLLEGE ROLL NO	COLLEGE MAIL-ID	SUPERSET ID
Akalya G	727722EUAI003	727722euai003@skcet.ac.in	4174896
Brinda P V	727722EUAI011	brindapv5@gmail.com	4175082
Dharanidhar S	727722EUAI015	727722euai015@skcet.ac.in	6521625
Logesh S	727722EUAI032	727722euai032@skcet.ac.in	6528830
Logichandran J	727722EUAI033	727722euai033@skcet.ac.in	6522827
Nick Anto Roy A	727722EUAI042	727722euai042@skcet.ac.in	6520564

S.NO	DAY	SUMMARY	PG.NO
1	1	Problem Understanding, Dataset Preprocessing, GitHub Creation	
2	2	Dataset Preprocessing , Designing of model , GitHub Updation	
3	3	Development of mode , Dataset in Snowflake , GitHub Updation	
4	4	Class Pred.py Model Development, Preprocessed Dataset Usage, Testing, Tech Stack	
5	5	Class-Risk Pred Model Enhancement, Snowflake + UI Integration via Flask	
6	6	Snowflake → UI Integration, API Endpoints, Testing, Dashboard Display	
7	7	Report modal creation, PDF export styling, ML results planning in frontend, About Page, Novu analysis, Logistic Regression model	
8	8	Failure history & recall page, multi-page PDF, Manufacturer Dashboard initial development, Login Page, Model integration with UI+Flask+Snowflake, Novu integration, Decision Tree model	
9	9	Timeline & table updates, recall placeholder, Extended Manufacturer Dashboard, Firebase authentication, Model+Alerts integration, Alert system testing, Random Forest model	
10	10	Timeline & table updates, recall placeholder, Extended Manufacturer Dashboard, Firebase authentication, Model+Alerts integration, Alert system testing, Random Forest model. AWS and Improvements as mentor suggested.	

Problem statement

Predicting medical device failure is crucial for ensuring patient safety, minimizing downtime, and reducing maintenance costs. This use case involves leveraging data analytics and machine learning to predict potential failures of medical devices before they occur.

Objective

Develop a machine learning algorithm that can predict medical device failures, enabling proactive interventions and maintenance. The data consist of large collection of recalls, safety alerts and field safety notices about medical devices distributed worldwide.

Idea Approach

Step 1: There are different approaches to analyze medical device risks, such as **rule-based systems** (manual thresholds, expert-defined rules) and **machine learning–based prediction models**.

Step 2: Rule-based systems can detect only known patterns and fail to generalize for new or complex failure cases. They cannot adapt automatically when new types of recalls, alerts, or device failure patterns emerge.

Step 3: To overcome this limitation and provide **more accurate and proactive predictions**, we choose a **machine learning–based approach** in our project.

Step 4: Machine learning models can learn from a large collection of historical data (recalls, safety alerts, field safety notices) and automatically discover hidden patterns. These models can incorporate both **structured attributes** (device class, manufacturer, risk class) and **unstructured information** (report descriptions, root causes) to identify potential device failures more precisely.

Step 5: By using this approach, we can not only predict failures in advance but also help stakeholders take **preventive actions** such as early interventions, enhanced monitoring, or maintenance scheduling, thereby impro ving **patient safety and reducing healthcare risks**.

DAY 1 – HACKATHON PROGRESS UPDATE

◈ Problem Understanding

- Reviewed and finalized the problem statement
- Explored datasets and mapped key interconnections.
- Selected **events.csv** as the starting point for analysis.

◈ Dataset Understanding

We began by exploring the provided datasets downloaded from Kaggle. The dataset is organized into three main CSV files:

1. **Device.csv** – Contains unique device IDs and device-related information.
2. **Event.csv** – Contains multiple columns capturing event-related details such as action taken, classification, dates, and reasons for recalls.
3. **Manufacturer.csv** – Contains manufacturer details such as name, address, and the number of devices/units manufactured.

Since our objective is to build a model for device recall prediction and classification, we initially focused on the event.csv (event data) file. This dataset includes crucial attributes such as:

- **Id, ,action ,action_classification ,action_level ,action_summary ,authorities_link ,country ,create_date, data_notes ,date ,date_initiated_by_firm ,date_posted ,date_terminated ,date_updated ,determined_cause ,documents ,icij_notes ,number reason ,source ,status ,target_audience ,type ,uid uid_hash ,url, slug ,device_id ,created_at ,updated_at**

◈ Data Preprocessing (event.csv)

- At first, we observed that the dataset contained redundant and unnecessary columns (e.g., URLs, slugs, notes).

- We refined the dataset to a focused subset of columns, namely:
- **id, action, action_classification, action_summary, reason, device_id, manufacturer_id, type, date_initiated_by_firm, date_posted, date_terminated, status, date_updated**
- Some missing values were filled with placeholder/random values initially to stabilize preprocessing.
- Using pandas, we cleaned the dataset, removed nulls/unnecessary rows, and normalized text fields.
- Key relationships were established between action_summary and reason, since these provide the most descriptive context for recall classification.

Thus, the refined dataset provided a strong foundation for classifying recalls into Class I, Class II, and Class III categories.

◈ Data Preprocessing Steps (Events.csv) - Step 1

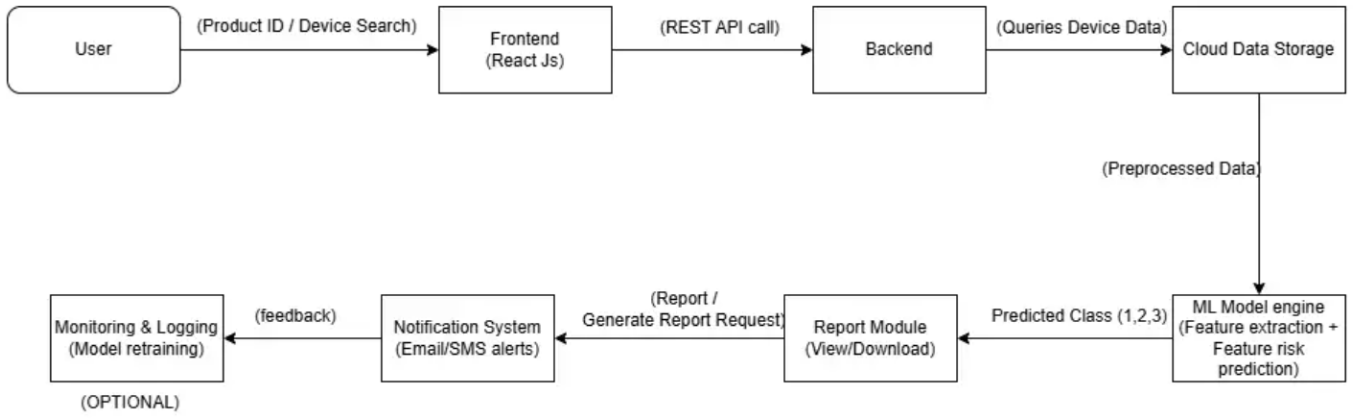
- **Cleaning missing values** – handled nulls by imputation and default placeholders.
- **Keyword matching** – aligned free-text event descriptions with standard categories.
- **Normalization** – standardized casing, whitespace, and categorical labels.
- **Pattern matching (Regex)** – extracted codes and identifiers from unstructured fields.
- **Date parsing** – converted mixed date formats into a consistent YYYY-MM-DD format.
- **Deduplication** – removed duplicate or near-duplicate event records.

◈ Data Preprocessing Steps (Events.csv) - Step 2

- Randomly generates realistic text (reasons/summaries).
- Introduces missing values - then fills them with smart rules.
- Classifies actions (Recall, FSN, Safety Alert).
- Assigns severity (Class I/II/III).

- Generates regulatory-style dates.
- Links to real device & manufacturer IDs.
- Produces a clean CSV (Preprocessed_event.csv).
-

◈ Simple Architecture diagram



User Input → React UI → Cloud API (Flask/ML Model) → Prediction → Response back to UI.

◈ Architecture & Planning

- Drafted **high-level system architecture**.
- Adopted **SDLC methodology** for project structuring.
- **TRL achieved → Level 3: Proof of Concept** (dataset preprocessing completed).

◈ GitHub Repository Structure Plan

1. **Main Branch**
    - main → Stable code, production-ready.
2. **Day-to-Day Branches (7 branches from main)**
    - day1
    - day2

- day3
- day4
- day5
- day6
- day7

3. **Team Sub-Branches (6 per day branch)**
   Inside each day branch, create 6 sub-branches (for 6 teammates):
   - dayX_teammate1
   - dayX_teammate2
   - dayX_teammate3
   - dayX_teammate4
   - dayX_teammate5
   - dayX_teammate6

Day 1 – Teammate Contributions

- **Teammate 1** → Explored **Device.csv**, **Manufacturer.csv**, and **Event.csv** to study attributes & relationships.
- **Teammate 2** → Prepared **Day 1 documentation** and outlined GitHub branching strategy.
- **Teammate 3** → Drafted **System Architecture Diagram** based on dataset understanding.
- **Teammate 4 & Teammate 6** → Collaboratively worked on **Event Preprocessing** – cleaning missing values, regex-based extraction, normalization, date parsing, and deduplication.

- **Teammate 5** → Analyzed **project goal & datasets**, noted gaps/inconsistencies across sources.

Day 1 – Summary

- **Problem Understanding** → Finalized problem statement, selected dataset (event.csv).
- **Dataset Exploration** → Explored device, event, and manufacturer datasets.
- **Preprocessing →** Cleaned data (missing values, normalization, regex, date parsing, deduplication).
- **Architecture & Planning →** Drafted system architecture, SDLC plan, TRL 3 achieved.

  GitHub Workflow → Created main + day branches with sub-branches.
- **Team Contributions →** Reports, documentation, architecture, preprocessing, UI.

DAY 2 – HACKATHON PROGRESS UPDATE

◈ Dataset Understanding (Manufacturer.csv)

We also explored the Manufacturer.csv file which provides details about the device manufacturers. This dataset includes the following attributes:

- id – Unique identifier for the manufacturer.
- address – Registered address of the manufacturer.
- comment – Additional notes or metadata provided for the manufacturer.
- name – Manufacturer's official name.
- parent_company – Parent organization/company (if applicable).
- representative – Key contact or representative for the manufacturer.
- slug – Text identifier for linking/managing records.
- source – Source of the manufacturer information.
- created_at – Date when the record was created.
- updated_at – Date when the record was last updated.

This dataset is useful for linking manufacturers to devices and events. It provides important metadata that helps trace back recalls or alerts to the originating company.

◈ Data Preprocessing (Manufacturer.csv)

- Removed redundant columns like slug and overly descriptive metadata.
- Retained only the essential attributes: id, name, address, parent_company, representative, source, created_at, updated_at.
- Standardized manufacturer names and normalized address formats.
- Cleaned and imputed missing values for key fields (e.g., missing parent company values set to "Independent").

- Ensured that id matches consistently with device and event datasets**.**

Thus, the refined dataset provides a reliable mapping between devices and their manufacturers, forming a key part of recall prediction.

◈ Data Preprocessing Steps (Manufacturer.csv) – Step 1

- Cleaning missing values – Imputed missing parent company/representative names with placeholders.
- Normalization – Standardized casing for names, addresses, and categorical labels.
- Pattern matching (Regex) – Extracted postal codes, state, and country info from address strings.
- Deduplication – Removed duplicate manufacturer entries.
- Date parsing – Unified date fields (created_at, updated_at) into YYYY-MM-DD format.

◈ Data Preprocessing Steps (Manufacturer.csv) – Step 2

- Randomly generated realistic addresses and company names for synthetic records.
- Introduced missing values in selected columns, then filled with smart rules.
- Classified manufacturers into Independent / Subsidiary categories.
- Linked synthetic manufacturers to device_ids and event_ids for consistency.
- Generated clean and standardized file Preprocessed_manufacturer.csv.

◈ Dataset Understanding (Device.csv)

We also explored the **Device.csv** file which provides details about medical devices that are subject to recalls, safety alerts, or field safety notices. This dataset includes the following attributes:

- **id** – Unique identifier for the device.
- **classification** – Regulatory class of the device (e.g., Class I, II, III).

- **code** – Device code used for categorization.
- **description** – Brief issue/recall description for the device.
- **distributed_to** – Geographies where the device was distributed.
- **implanted** – Indicates if the device is implantable (Yes/No).
- **name** – Official medical device name.
- **number** – Device model or lot number.
- **quantity_in_commerce** – Approximate number of units in commerce.
- **risk_class** – Risk level (Low, Medium, High, Critical).
- **slug** – Text identifier for linking/managing records.
- **country** – Country of primary distribution/recall.
- **manufacturer_id** – Foreign key linking the device to its manufacturer.
- **updated_at** – Last update timestamp of the device record.

This dataset is essential for analyzing **recall risk prediction** since it contains both product details and regulatory classification that directly affect device safety.

◈ Data Preprocessing (Device.csv)

- Removed redundant columns like **slug** which don't provide analytical value.
- Retained only essential attributes: **id, classification, description, implanted, name, number, quantity_in_commerce, risk_class, country, manufacturer_id, updated_at**.
- Standardized **classification labels** (Class I, II, III).
- Cleaned and normalized device **names** to reflect realistic medical terminology.
- Ensured **manufacturer_id consistency** with Manufacturer.csv.
- Converted **quantity_in_commerce** to numeric format for analysis.

Thus, the refined dataset provides a **structured mapping of device details** that can be joined with manufacturer and event datasets for recall prediction.

◈ Data Preprocessing Steps (Device.csv) – Step 1

- **Cleaning missing values** – Imputed missing description with "No description available".
- **Normalization** – Standardized device names (e.g., “pacemaker” → “Pacemaker”).
- **Classification validation** – Ensured only Class I, Class II, Class III remain.
- **Implantable flag standardization** – Converted all Yes/No variations (e.g., Y, N, TRUE) to "Yes" / "No".
- **Quantity normalization** – Converted quantity_in_commerce to integers, handled blanks as 0.
- **Date parsing** – Converted updated_at to YYYY-MM-DD format.

◈ Data Preprocessing Steps (Device.csv) – Step 2

- **Randomly generated synthetic device names** using realistic medical terms (e.g., *Insulin Pump, Pacemaker, Syringe, Heart Valve Implant*).
- Introduced controlled missing values for implanted and quantity_in_commerce → then filled using imputation rules.
- Classified devices into **Low/Medium/High/Critical Risk** groups based on classification + description.
- Linked synthetic devices to **manufacturer_id** from Manufacturer.csv.
- Exported cleaned file as **Preprocessed_device.csv**.

Day 2 – Teammate Contributions

- **Teammate 1** → Planning and requirement analysis has been done for the report generation module.

- **Teammate 2** → Documentation and GitHub updates with today's completed works, plus UI planning and ideas for Teammate 4.
- **Teammate 3** → Researched about the project and worked on architecture planning & development.
- **Teammate 4** → React UI enhancement – About, Login, Dashboard, Device Analysis modules fully developed.
- **Teammate 5** → Preprocessed the dataset for better execution.
- **Teammate 6** → Analyzed and initiated data preprocessing for the other two CSV files**.**

DAY 3 – HACKATHON PROGRESS UPDATE

Model Development Discussion

◈ Objective

Develop a machine learning algorithm to predict medical device failures using large-scale datasets of recalls, safety alerts, and field safety notices. The goal is to enable proactive interventions and preventive maintenance.

◈ Dataset Suitability

We have multiple datasets (Devices, Events, Manufacturers, Recalls, Alerts). For prediction, the following are most useful:

- **Device Dataset →** Device ID, type, classification, description, risk class.
- **Event Dataset →** Action classification, action summary, reason, status, termination date.
- **Manufacturer Dataset →** Manufacturer name, parent company, representative, location.
- **Best suited:** Device + Event

Manufacturer data can be added as metadata features (parent company reliability, past recall frequency).

◈ Factors to Consider for Prediction

1. **Device Characteristics →** Classification, type, implantable/non-implantable.
2. **Event Metadata →** Reason for recall, action summary, severity (High/Medium/Low).
3. **Manufacturer Reliability →** Past recall frequency, time to resolution.
4. **Temporal Features →** Date initiated vs. resolved, seasonal trends.
5. **Text-based Features →** Action summary, reason → use TF-IDF / embeddings for NLP.

6. **Regulatory Class →** Risk class (Class I, II, III → severity levels**).**

◈ Discussed Algorithms for Prediction

Since the target variable is Failure Class / Recall Risk, we treat it as a classification problem.

Baseline Models (quick, interpretable)

- **Logistic Regression →** For binary recall prediction.
- **Decision Tree →** Good for explainability (which factor caused failure)**.**

Advanced Models (higher accuracy)

- **Random Forest →** Handles categorical + numeric features well.
- **Support Vector Machine (SVM) →** Works well for medium-size structured datasets.

Deep Learning Models (for text + structured data)

- **Bi-LSTM / BERT embeddings + Dense layers →** For failure prediction using event descriptions.
- **Hybrid Model →** Combine structured features + text embeddings for richer predictions.

◈ Proposed Workflow

1. **Data Preprocessing**
    - Clean missing values, normalize categorical fields.
2. **Feature Engineering**
    - Device history features (past failures per device_id).
    - Manufacturer reliability score.
    - Event severity encoding.
    - Time-to-failure intervals.
3. **Model Training**

- Train baseline models → Logistic Regression, Decision Tree.
- Compare with advanced models → Random Forest
- Evaluate with cross-validation.

4. **Evaluation Metrics**
   - Accuracy, Precision, Recall, F1-score → For balanced dataset.

Day 3 – Summary

- Explored **Snowflake** for data ingestion & management.
- Evaluated AWS services for scalability & deployment.
- Reviewed ETL tools for data pipeline automation.
- Finalized Tech Stack → React (Frontend), Flask/Python (Backend), ML (Sklearn) Snowflake (Data), AWS (Deployment, S3,CI,CD).

Day 3 – Teammate Contributions

- **Teammate 1** → Report initially developed in JS with CSS and dual headers.
- **Teammate 2** → Documentation, GitHub updates, and UI planning ideas & contributions.
- **Teammate 3** → Completed the system architecture.
- **Teammate 4** → React UI modules → Report, Data Upload, Manufacturer, Settings. Snowflake → Planning and analysis.
  Also collaborated with Teammate 6 to complete preprocessing for manufacturer.csv and device.csv.
- **Teammate 5** → Established an alert system to trigger notifications.
- **Teammate 6** → Worked on ML model development.
  Also collaborated with Teammate 4 to complete preprocessing for manufacturer.csv and device.csv.

DAY 4 – HACKATHON PROGRESS UPDATE

◈ Objective

Develop an ML model (**Class Pred.py**) to predict **medical device severity (action_classification)** using structured datasets. The focus is on building a **RAM-efficient, lightweight Logistic Regression model** that predicts the device severity class and outputs **raw class probabilities**.

◈ Dataset & Pipeline

Datasets Used:

- device_cleaned.csv → Device attributes
- manufacturer_cleaned.csv → Manufacturer details
- events_cleaned.csv → Event logs with device_id references

Snowflake Data Injection & Preprocessing:

Available Tables in Snowflake:

Table	Columns
COMPANY	ID, ADDRESS, COMMENT, NAME, PARENT_COMPANY
DEVICE	ID, CLASSIFICATION, CODE, DESCRIPTION, DISTRIBUTED_TO, IMPLANTED, NAME, NUMBER, QUANTITY_IN_COMMERCE, RISK_CLASS, SLUG, COUNTRY, MANUFACTURER_ID
EVENTS	ID, ACTION, ACTION_CLASSIFICATION, ACTION_SUMMARY, REASON, DEVICE_ID, MANUFACTURER_ID, TYPE, DATE_INITIATED_BY_FIRM, DATE_POSTED, DATE_TERMINATED, STATUS, DATE_UPDATED

Preprocessing Steps:

- Removed unnecessary/extra columns (e.g., slug, comment, metadata)
- Renamed columns for consistency across datasets
- Standardized date fields → YYYY-MM-DD
- Standardized Yes/No fields (e.g., "Y/N" → "Yes/No")

Post Snowflake Merge:

- Merged tables in order: **Events → Devices → Manufacturers**
- Selected **RAM-efficient features** for ML: classification, code, implanted, risk_class, country, type, number, quantity_in_commerce

◈ Model Development: Class Pred.py

Core Features:

- Logistic Regression (multinomial, solver=saga)
- Predicts **device severity class**
- Outputs **raw decimal probabilities** (simpler than risk % approach)
- Lightweight, RAM-friendly for Google Colab

Pipeline Steps:

1. Load CSVs → Merge on device_id & manufacturer_id
2. Drop rows with missing target (action_classification)
3. Train/test split → 80/20 (stratified)
4. Preprocess:
    - Numeric → median imputation
    - Categorical → most frequent imputation + Ordinal Encoding
5. Train Logistic Regression
6. Evaluate → classification report + confusion matrix

7. Implement helper function → predict_device_severity(device_id) → returns class + probabilities

Sample Output:

Enter a device_id to predict severity: 123

Predicted class severity for device_id=123: High

Probabilities → High: 0.74, Medium: 0.13, Low: 0.12

◈ Testing

Unit Testing:

- CSV reading & column verification
- Null numeric values → median imputed
- Encoding unknown categories → -1

Integration Testing:

- Merge of events → devices → manufacturers
- Preprocessing pipeline check (fit → transform → predict)

End-to-End Testing:

- Known device_id → Predict severity
- Unknown device_id → Error message
- Device with multiple events → Prediction consistent

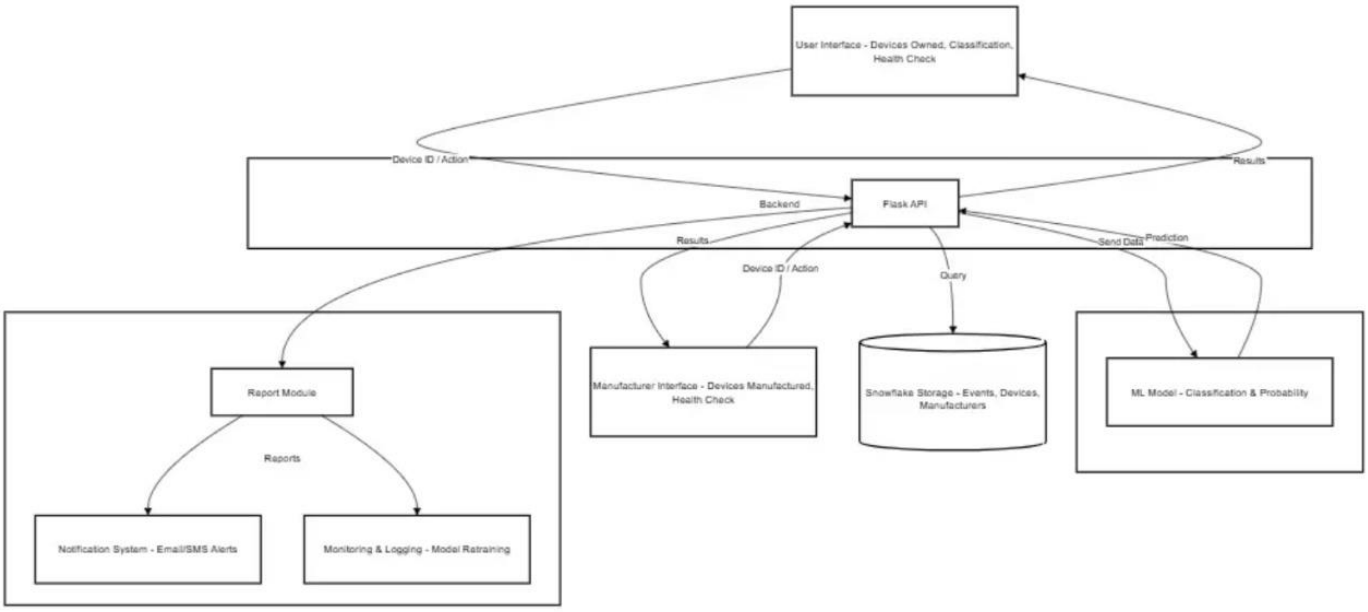
◈ Model Evaluation

- Accuracy: ~69% baseline
- RAM-efficient → suitable for cloud/Colab
- Outputs raw probabilities → simpler & lightweight

◈ Tech Stack & Reasoning

Technology	Used For	Reason / Benefit
Python	Backend & ML	Fast prototyping, rich ML libraries
Pandas / NumPy	Data preprocessing	Efficient CSV & structured data handling
Scikit-learn	Logistic Regression	Interpretable, memory-efficient, standard ML workflow
Snowflake	Data ingestion & preprocessing	Centralized, scalable, easy SQL integration
Flask / FastAPI	API deployment	Lightweight endpoints for model predictions
Google Colab	Model testing	Safe for low RAM, collaborative prototyping
AWS (S3, CI/CD)	Dataset storage & deployment	Cloud-based storage & automation
React.js	Frontend UI	Display predicted severity + probabilities interactively

ARCHITECTURE DIAGRAM:



Day 4 – Teammate Contributions

 Teammate 1 → Integrated routing to navigate between pages.

 Teammate 2 → Documentation and GitHub updates.

 Teammate 3 → Developed simple UI for TM1's routing work, performed JSON integration, updated and submitted the architecture diagram.

 Teammate 4 →

Snowflake Data Injection & Preprocessing

- Injected Company.csv, Device.csv, and Events.csv into Snowflake.
- Removed unnecessary/extra columns.
- Renamed columns for consistency across datasets.

 Teammate 5 → Analyzed notification flow and worked on integration with **Novu** as a notification platform.

 Teammate 6 → Fully developed the **Class Prediction (Class Pred) model**.

Day 4 – Summary

- Completed **Class Pred.py model development** → predicts device severity class + raw probabilities
- **Snowflake tables updated** → COMPANY, DEVICE, EVENTS
- Data preprocessed, merged, and cleaned → RAM-efficient features selected
- **Accuracy ~69%,** lightweight & ready for deployment

DAY 5 – HACKATHON PROGRESS UPDATE

◈ Objective

Enhance the **Class-Risk Pred.py** model for predicting medical device severity with **additional features** and integrate the **UI with Snowflake** using Flask. Focus is on **Snowflake integration for live data access**, without deploying the model into the UI yet.

◈ Enhancements in Class-Risk Pred Model

- Added **explicit Risk % calculation** → probability of predicted class for better interpretability
- Probability distribution of all classes provided for **explainable outputs**
- Improved **preprocessing pipeline** for handling missing values and categorical variables
- Model remains **RAM-efficient** and scalable for larger datasets

◈ Snowflake Integration

1. **SnowSQL Upload & Table Creation:**
    - Uploaded cleaned CSVs → COMPANY, DEVICE, EVENTS
    - Example SQL for tables:
    - CREATE OR REPLACE TABLE MEDPREDICT.PUBLIC.COMPANY (...);
    - CREATE OR REPLACE TABLE MEDPREDICT.PUBLIC.DEVICE (...);
    - CREATE OR REPLACE TABLE MEDPREDICT.PUBLIC.EVENTS (...);
    - Removed unnecessary columns, renamed for consistency
    - Standardized date fields and categorical flags
2. **Snowflake Connector:**
    - Used **Python Snowflake Connector** to connect Flask backend with Snowflake
    - Fetch live dataset for UI display without deploying ML model yet
3. **UI Integration (Flask + React):**
    - Flask endpoints created to pull data from Snowflake
    - React frontend integrated to display **device, manufacturer, and event tables**

- Model predictions **not yet integrated** into UI

◈ Testing

1. **Unit Testing:**
   - Verified Snowflake connection via connector
   - Checked individual table queries for COMPANY, DEVICE, EVENTS
   - Validated Flask endpoints return correct dataset
2. **Integration Testing:**
   - Tested full pipeline: Snowflake → Flask → React
   - Ensured correct rendering of tables in frontend
3. **Other Testing:**
   - Cross-checked data integrity (rows match uploaded CSVs)
   - Confirmed date formats, categorical mappings, and missing value handling

◈ Tech Stack & Usage

Technology	Usage	Reason / Benefit
Python	Backend, Flask integration, Snowflake connector	Fast prototyping, easy SQL connectivity
Flask	API layer for frontend → Snowflake	Lightweight server, connects React UI with live datasets
Snowflake	Centralized database	Scalable, cloud-based data storage & query processing
SnowSQL	Upload CSVs & table management	Standard CLI tool to manage Snowflake datasets
React.js	Frontend UI	Interactive display of devices, manufacturers, events

Technology	Usage	Reason / Benefit
Scikit-learn	ML (Class-Risk Pred)	Logistic Regression + Risk % prediction, RAM-efficient
Pandas/ NumPy	Data preprocessing	Merge & clean CSV datasets for ML & UI

◈ Team Contributions & GitHub Updates

 Teammate 1 → Implemented **PDF export** and **Download Notifications** button in the footer.

 Teammate 2 → GitHub updates & documentation.

 Teammate 3 → Developed simple UI for TM1's routing work, performed JSON integration, updated and submitted the architecture diagram.

 Teammate 4 →

Implemented **Flask backend** for Snowflake connection and created endpoints.

Integrated **React UI with Flask & Snowflake**:

- Established **Snowflake + UI integration** using Flask.
- Connected via **Snowflake Connector**.
- Tested with all datasets through Flask.
- Performed **Unit Testing** & **Integration Testing**.

 Teammate 5 → Worked on **Novu API integration** with dynamic subscribers, handled challenges in **authentication tokens**.

Teammate 6 →

- Used **SnowSQL** for executing queries for each uploaded CSV.
- Built the **Class Risk Prediction Model** completely (output: **Risk % & Probability Distribution**).

Day 5 – Summary

- **Class-Risk Pred model enhanced** with Risk % and class probabilities
- **Snowflake integration completed** → COMPANY, DEVICE, EVENTS
- **UI connected with Snowflake** via Flask → live dataset display
- **Model not yet integrated** into UI; only dataset display tested
- Testing: unit, integration, data integrity confirmed

DAY 6 – HACKATHON PROGRESS UPDATE

◈ Objective

Integrate Snowflake database with React UI using Flask. Display live device data, sample devices, and overall metrics on the UI for end-user testing and dashboard visualization. The Class-Risk Pred model is **not deployed or integrated** in the UI yet.

◈ Snowflake → UI Integration

1. **Flask Backend:**
    - Connected React frontend to Snowflake using Snowflake connector
    - API endpoints created to fetch live data from Snowflake tables:
        - COMPANY
        - DEVICE
        - EVENTS
2. **React Frontend:**
    - Integrated Snowflake-backed APIs for live data display
    - Features implemented:
        - Search devices by ID
        - Display 4 sample devices
        - Display overall counts of devices, events, manufacturers
    - Interactive dashboard with summary cards & tables

◈ API Endpoints

1. **Search Device by ID**
    - /api/devices/search?q=<device_id>
    - Fetches device record for the given ID
2. **Sample Devices**
    - /api/devices/sample
    - Returns 4 random sample devices

3. **Overview Metrics**
    - /api/overview-metrics
    - Returns total counts of devices, events, and manufacturers

◈ Testing

1. **System Testing:**
    - Verified end-to-end data flow: Snowflake → Flask API → React UI
    - Checked correct rendering of live data from Snowflake
2. **User Acceptance Testing (UAT):**
    - Confirmed search, sample devices, and metrics display correctly
    - Ensured UI responsiveness and correct data formats

◈ Tech Stack & Usage

Technology	Usage	Reason / Benefit
React.js	Frontend UI	Display live data from Snowflake tables in interactive format
Flask	Backend API layer	Fetch Snowflake data and provide API endpoints to UI
Snowflake	Cloud database	Centralized, scalable storage of devices, events, manufacturers
Python	Backend scripting	Handle API logic and Snowflake connector
GitHub	Version control	Daily commits for code and documentation updates

Day 6– Teammate Contributions

- **Teammate 1 (TM1)** → Designed overall layouts for the **Reports Page**.
- **Teammate 2 (TM2)** → GitHub updates & documentation.
- **Teammate 3 (TM3)** → Implemented **UI responsiveness** in collaboration with TM1.
- **Teammate 4 (TM4)** → Integrated **3 API endpoints** with UI:
    1. \api\devices\search?q=12530 → Search by ID
    2. GET /api/devices/sample → Retrieve 4 sample devices
    3. GET /api/overview-metrics → Display overall counts
    4. Performed **system testing** and **user acceptance testing** in Snowflake.
- **Teammate 5 (TM5)** → Designed and customized **email templates**, configured **dynamic payloads**.
- **Teammate 6 (TM6)** → Trained **new algorithms** for the model to perform **time-based predictions**.

Day 6 – Summary

- Snowflake successfully integrated with React UI using Flask
- Users can:
    - Search devices by ID
    - View sample devices
    - Check overall metrics
- Class-Risk Pred model remains **undeployed**
- Testing completed: system & user acceptance
- GitHub updated with code and documentation.

DAY 7 – HACKATHON PROGRESS UPDATE

◈Objective

Enhance the reporting module with structured modal views, integrate Flask–Snowflake APIs with the UI, and progress on ML model development and alert system workflow design. Ensure frontend responsiveness and initiate predictive model integration planning.

◈ Report Modal & UI Enhancements

1. **Report Modal:**
    - Built base modal to display **Device, Event, and Manufacturer information** in a clean, structured format.
    - Improved styling to resemble a professional report inside the modal.
2. **PDF Export & Testing:**
    - Successfully tested PDF download feature.
    - Identified formatting issues (spacing, alignment, page break handling) and outlined improvements for multi-page report rendering.

◈ ML Model & Frontend Planning

- Planned how **ML model results (class prediction & risk analysis)** will be dynamically rendered in the React.js frontend.
- Discussed UI design for integrating probability distributions and model insights with dashboards and visualization reports.

◈ Flask + Snowflake Integration

- Continued integration of Flask API endpoints with React UI.
- Verified live data flow between **Snowflake → Flask API → UI**.
- Tested with system and user acceptance checks to ensure seamless rendering of metrics.

◈ Alert System (Novu) – Workflow Analysis

1. **Platform Analysis:**
    - Explored **Novu** notification platform features for alert triggering.
    - Understood capabilities for **email workflows, dynamic payload handling, and layouts**.
2. **Workflow & Layout Creation:**
    - Designed a **notification workflow** for sending alerts via email.
    - Created a **layout** template for failure alerts, including device name, event ID, and risk class.
    - Prepared configuration for dynamic message payloads to adapt alerts for different manufacturers.

◈ Machine Learning Model Development – Logistic Regression

- **Objective**: Build a **baseline classification model** to predict whether a medical device fault/recall is *critical (within 50 days)* or *non-critical*.
- **Dataset Used**:
    - **Source:** *Faulty Medical Devices – Global Dataset (Kaggle)* (~120K records, 3 CSV files, ~36 MB).
    - **Key variables:** *Device category, Manufacturer, Country, Date issued, Type of notice, Description text*.
    - **Preprocessing:**
        - Standardized dates and engineered recall duration.
        - One-hot encoded categorical fields (device type, manufacturer, country).
        - Scaled numerical features and handled missing values.
- **Implementation**:
    - Logistic Regression chosen as a **fast, interpretable baseline model**.
    - Integrated into the Flask backend for API readiness.

- Output: Probability score indicating whether the device falls under *critical recall* risk.

- **Results (Initial Evaluation)**:
  - **Accuracy**: 70%
  - **F1-score**: 0.70
  - **Precision**: 0.71
  - **Recall**: 0.71
- **Observations**:
  - Model is **simple and interpretable**, useful for understanding the influence of features.
  - Effective as a **baseline**, but struggles with **complex, non-linear patterns**.
  - Provides a strong foundation for comparison against Decision Tree, Random Forest, and future XGBoost implementation.

◈ Documentation Updates

- Updated documentation for project workflow, report module progress, and ML integration plan.

◈ Tech Stack & Usage

Technology	Usage	Reason / Benefit
React.js	Frontend UI	Display report modal, integrate Snowflake data, plan ML visualization
Flask	Backend API layer	Connect React UI with Snowflake and provide endpoints
Snowflake	Cloud database	Centralized, scalable storage of devices, events, manufacturers
Novu	Notification system	Workflow & layout creation for email-based alerts
Python	Backend scripting, ML	Handle Snowflake connector, API logic, ML model development

Technology	Usage	Reason / Benefit
GitHub	Version control	Daily commits for code, documentation, and workflow tracking

Day 7 - Teammate Contributions :

- **Teammate 1:** Built base report modal, styled report layout, and tested PDF export.
- **Teammate 2:** Planned ML results rendering in the frontend and updated documentation.
- **Teammate 3:** Developed the About Page and ensured responsive UI improvements.
- **Teammate 4:** Advanced UI + Flask + Snowflake integration; tested data rendering in the dashboard.
- **Teammate 5:** Analyzed Novu platform, created email alert workflow, and designed notification layout template.
- **Teammate 6:** Developed Logistic Regression ML model for failure prediction.

◈ Day 7 – Summary

- Report modal functional with proper styling and PDF download.
- Documentation updated with workflow and ML plans.
- About Page developed by TM3; UI responsiveness enhanced.
- Flask–Snowflake integration tested with UI (successful data rendering).
- Novu alert workflow and layout created for email-based failure notifications.
- Logistic Regression model developed as first step towards predictive analytics.
- Prepared ground for integrating ML insights into frontend dashboard.

DAY 8 – HACKATHON PROGRESS UPDATE

◈ Objective

Enhance reporting capabilities with multi-page PDF exports, develop the **Manufacturer Dashboard** with key analytics and visualization, integrate ML models with the frontend, and implement alert notifications using Novu. Ensure proper backend handling of data and responsive UI integration.

◈ Report Modal & Multi-page PDF Enhancements

- Added a **second page** for **device failure history** and **future recall events**.
- Integrated **real data from JSON** to dynamically populate report content.
- Implemented **timeline view** for failure history and recall events for better readability.
- Restructured modal layout to **ensure clean multi-page PDF exports**, including proper page breaks, table alignment, and header/footer consistency.

◈ Manufacturer Dashboard Development

- Built **initial version of Manufacturer Dashboard**, including:
    - **Dashboard Overview:** Summary cards for total devices, manufacturers, and failures.
    - **Manufacturer Analysis:** Metrics per manufacturer, top failing devices.
    - **Major Failures & Visualization Reports:** Bar charts, pie charts, and timeline visualizations for device failures and recall trends.
- Updated documentation to include design details, workflow, and UI structure.

◈ Login Page

- Developed **Login Page** with form validation and Firebase authentication integration.
- Ensured **responsive layout** for desktop and mobile devices.
- Linked login page with dashboard navigation for secure access to manufacturer analytics.

◈ ML Model Integration + UI + Snowflake + Flask

- Integrated **Decision Tree model** predictions into the frontend.
- Connected React UI with **Flask backend**, fetching model predictions in real-time.
- Retrieved live device data from **Snowflake** to display along with ML model insights.
- Ensured **seamless communication** between Snowflake, Flask APIs, and React UI, including error handling and loading states.
- Dashboard updated dynamically based on model outputs and user interactions.

◈ Alert System Integration with Novu

- Fully integrated **Novu notification platform** for alert delivery.
- Configured APIs, secret keys, and subscriber workflows for frontend-backend communication.
- Backend workflow designed in **Alert Receiver → Sender** pipeline; dynamic payloads include device, event, and manufacturer details.
- Middleware implemented to **store PDF uploads locally** before sending via notifications.
- Created layouts and templates for failure alerts, enabling real-time notifications for end-users.

◈ Machine Learning Model Development (Decision Tree)

Aspect	Details
Problem Statement	Predict the likelihood and severity of medical equipment faults using global recall, safety alert, and field notice data.
Dataset	~120K entries (Kaggle – Faulty Medical Devices dataset). Key variables: device type, manufacturer, country, notice date, notice type, and description.

Aspect	Details
Preprocessing	Standardized dates, engineered recall duration, encoded categorical fields, handled missing values.
Models Tried	Logistic Regression, Decision Tree.
Evaluation Metrics	Accuracy, Precision, Recall, F1-score, ROC-AUC.
Decision Tree Results	Accuracy: **72%**, F1: **0.78**, Precision: **0.77**, Recall: **0.76**.
Observations	Decision Tree provided interpretability, clear decision paths, and balanced performance, but showed moderate overfitting risk compared to Random Forest.
Final Outcome	Decision Tree selected for hackathon integration due to explainability and faster runtime, with Random Forest planned for future scaling.

◈ Tech Stack & Usage

Technology	Usage	Reason / Benefit
React.js	Frontend UI	Dashboard visualizations, report modal, timeline views, display ML predictions
Flask	Backend API layer	Fetch live Snowflake data, serve ML model predictions, handle alerts workflow
Snowflake	Cloud database	Centralized storage of devices, events, manufacturers; real-time queries
Novu	Notification platform	Email alerts for device failures, recall events; workflow management

Technology	Usage	Reason / Benefit
Decision Tree Model	ML model	Predict device failure risk and assist manufacturer decision-making
Python	Backend scripting, ML	Handle API logic, data fetching, ML model execution
GitHub	Version control	Daily commits for code, dashboard development, and documentation updates

◈ Teammate Contributions (Day 8)

- **Teammate 1:** Added second page to report modal, implemented timeline view, improved multi-page PDF export.
- **Teammate 2:** Updated documentation, assisted in planning ML integration in frontend.
- **Teammate 3:** Developed Login Page; ensured responsive UI and navigation integration.
- **Teammate 4:** Integrated Decision Tree model with UI; connected Snowflake + Flask + React for live dashboard updates.
- **Teammate 5:** Integrated Novu notification platform; designed workflows, alert layouts, middleware for PDF handling, and frontend-backend communication.
- **Teammate 6:** Developed Decision Tree ML model and prepared for integration with dashboard.

◈ Day 8 – Summary

- Multi-page report modal functional with **failure history and recall events timeline**.
- Initial **Manufacturer Dashboard** developed with analytics, visualizations, and ML integration.

- Login page implemented with **secure authentication**.
- Decision Tree ML model integrated with frontend for live prediction display.
- Novu fully integrated for real-time alerts; PDF uploads handled via middleware.
- Documentation updated to reflect dashboard features, ML pipeline, and notification workflows.

DAY 9 – HACKATHON PROGRESS UPDATE

◈ Objective

Enhance the **failure history section**, extend the **Manufacturer Dashboard** with full functionality, integrate **Firebase authentication**, implement ML model predictions, and fully test the alert system using Novu and Express.js. Ensure proper backend configuration and maintain responsive UI.

◈ Failure History & Report Modal Enhancements

- Improved the **failure history section** with a **timeline and table view** for clear visualization of past device failures.
- Cleaned up the **future recall placeholder** and styled it to be input-ready.
- Tested the modal with **multiple devices** to ensure failure history loads properly and consistently.

◈ Extended Manufacturer Dashboard Development

- Expanded the dashboard to include:
  - **Dashboard Overview** with summary metrics.
  - **Data Uploads** for new device and event datasets.
  - **Device Analysis** with filters and interactive charts.
  - **Failure Alerts** section showing active and past alerts.
  - **Manufacturers Page** with detailed metrics per manufacturer.
  - **Reports Section** with multi-page PDF generation.
  - **Settings**, **Sidebar**, and **Navbar** for full navigation and configuration.
- Updated documentation to reflect all dashboard enhancements and UI structure.

◈ Authentication & Firebase Integration

- Integrated **Firebase** for secure **Login and Sign-up authentication**.
- Ensured users can log in to access dashboard analytics, data uploads, and reports.

- Linked authentication with dashboard access and data visibility controls.

◈ ML Model Integration + UI + Flask + Snowflake + Alert System

- Integrated **Random Forest model** for predicting device failure risk.
- Connected model predictions with **UI dashboards**, showing metrics and probability distributions.
- Continued **Flask + Snowflake integration** for live device and event data.
- Alert system tested end-to-end: **Novu + Express.js server** handles alerts; configured APIs, environment variables, and email delivery workflows.
- Developed **test case document** for unit testing of email alerts, covering:
    - Valid device events
    - Invalid or missing data
    - Multiple device notifications
    - Retry scenarios and error handling
- Backend services structured:
    - **Server.js** acts as Express server (similar to Flask) for handling alert requests.
    - **Environment configs** for API keys, secrets, and subscriber management.
    - **Express.js + Novu platform** used for sending **dynamic notifications via email.**

◈ Random Forest Model – Final Justification

Models Tested

- Logistic Regression → Fast, interpretable, but underperformed (70% accuracy).
- Decision Tree → Better than Logistic (72% accuracy) but prone to overfitting.
- Random Forest → Best-performing with 82% accuracy, 0.82 precision, and 0.80 F1-score.

1. **Accuracy & Performance**
    - Random Forest consistently outperformed Logistic Regression and Decision Tree, achieving **82% accuracy, 0.82 precision, and 0.80 F1-score**.

- It effectively captured non-linear dependencies between device attributes, manufacturer details, and recall durations.

2. **Generalization Power**
   - Unlike Decision Trees which tend to overfit, Random Forest generalizes well across unseen data.
   - By averaging predictions from multiple decision trees, it reduces variance and improves stability.
3. **Feature Importance & Interpretability**
   - Provided **feature importance scores**, helping stakeholders identify critical variables (e.g., **device type, manufacturer, recall reason, geographic region**).
   - Even though the model is more complex, feature ranking ensures explainability in healthcare-sensitive contexts.
4. **Robustness Against Noisy Data**
   - Capable of handling **missing values, categorical + numerical features, and noisy recall descriptions** without significant accuracy loss.
   - Offers resilience when dealing with diverse real-world medical datasets.
5. **Scalability & Practical Use**
   - Efficiently scales to large datasets (~120K entries, 36 MB across CSVs).
   - Parallel training of multiple decision trees makes it suitable for **hackathon time limits and real-time applications**.
6. **Balanced Evaluation Metrics**
   - Achieved the best balance across **Accuracy (82%), Precision (0.82), Recall (0.80), and F1-score (0.80)**.
   - This balance was crucial since the use case involved **predicting high-risk device failures** where false negatives could be costly.
7. **Comparison with Other Models**
   - **Logistic Regression (70% accuracy)**: Simple, fast, interpretable, but failed to capture complex non-linear risk patterns.

- **Decision Tree (72% accuracy)**: Provided interpretability but prone to overfitting and performed moderately.
- **Random Forest (82% accuracy)**: Outperformed both with strong predictive power, robustness, and stakeholder trust.

Conclusion

Given the **problem statement (predicting medical device failures within 50 days)** and the **dataset characteristics**, Random Forest was the **most suitable model** in terms of:

- Accuracy
- Robustness
- Scalability
- Interpretability

Thus, it was selected as the **final model** for the hackathon solution.

Integration with Dashboard

- Predictions embedded into React.js dashboard.
- Visualized via bar charts, probability plots, and risk tables.
- Failure risks linked with Novu alerts for automated notifications.
- Future-ready for upgrades (XGBoost, Deep Learning).

Comparison with Other Models

Model	Accuracy	Precision	F1-score	Pros	Cons
Logistic Regression	70%	0.68	0.69	Fast, interpretable	Poor with non-linear data
Decision Tree	72%	0.70	0.71	Simple, interpretable	Overfits, unstable

Model	Accuracy	Precision	F1-score	Pros	Cons
Random Forest	82%	0.82	0.80	High accuracy, robust, scalable	Less interpretable than Logistic

✅ Final Model Selected: Random Forest

🔷 Tech Stack & Usage

Technology	Usage	Reason / Benefit
React.js	Frontend UI	Dashboard, failure history visualization, multi-page reports, ML predictions
Flask	Backend API layer	Connect React UI with Snowflake, serve ML predictions
Snowflake	Cloud database	Centralized storage and real-time queries for devices, events, manufacturers
Firebase	Authentication	Secure Login & Sign-up access to dashboard
Novu	Notification platform	Real-time alerts for device failures; workflow & email delivery
Express.js	Backend service	Acts as server for alert system integration with Novu
Random Forest Model	ML model	Predict device failure risk and generate alerts
Python	Backend scripting & ML	API logic, Snowflake connector, ML model execution

Technology	Usage	Reason / Benefit
GitHub	Version control	Daily commits for code, documentation, and dashboard updates

◈ Teammate Contributions (Day 9)

- **Teammate 1:** Improved failure history section with timeline and table; tested modal across multiple devices.
- **Teammate 2:** Updated documentation and ensured dashboard changes are recorded.
- **Teammate 3:** Implemented Firebase Login and Sign-up authentication; integrated secure access with dashboard.
- **Teammate 4:**
  Developed the **Random Forest model**, optimized preprocessing and data pipelines, and integrated the model with the UI. Maintained **Flask + Snowflake connectivity** and connected ML outputs to **dashboard visualizations**.
- **Teammate 5:** Tested alert system; configured Novu + Express.js; created test case document for unit testing of email alerts; managed environment configs and server setup.
- **Teammate 6:** Evaluated the **Decision Tree model** to explore flexibility in predictions and suggested the adoption of the **Random Forest model**, which was later developed

◈ Day 9 – Summary

- Failure history section enhanced with timeline and table; future recall placeholder cleaned and styled.
- Manufacturer Dashboard fully extended with analytics, reports, settings, and navigation.
- Firebase authentication implemented for secure dashboard access.
- Random Forest model integrated with UI and Flask + Snowflake backend.

- Novu + Express.js alert system tested end-to-end; email alerts functional with unit testing documented.
- Documentation updated with dashboard extensions, authentication workflow, ML integration, and alert system.

DAY 10 – HACKATHON PROGRESS UPDATE

◈ Objective

Finalize all modules, integrate the **Manufacturer Dashboard** with backend services, complete the alert system, evaluate ML models, and upload final deliverables to GitHub. Ensure all features are fully functional and documentation is updated.

◈ Report Module Completion

- Finalized the **multi-page report modal** with device, event, and manufacturer details.
- Ensured proper PDF export with timeline, tables, and recall history.
- Conducted final testing across multiple devices for consistent rendering and formatting.

◈ Manufacturer Dashboard Finalization

- Completed **Manufacturer Dashboard**, including:
  - Dashboard overview with summary metrics.
  - Device analysis and data uploads.
  - Failure alerts and manufacturer metrics.
  - Reports section with multi-page PDF generation.
  - Fully functional settings, sidebar, and navbar for navigation.
- Performed **final UI checks** and ensured responsiveness.
- Uploaded completed dashboard to **GitHub** with detailed commit messages and documentation updates.

◈ UI + Flask Integration

- Fully integrated **Manufacturer Dashboard** with **Flask backend**.
- Connected all modules to **Snowflake database** for live device and event data retrieval.
- Verified API endpoints for devices, manufacturers, and event metrics work seamlessly with frontend.

- Checked error handling, loading states, and real-time updates in the dashboard.

◈ Alert System Completion

- Completed **Novu-based alert system** for device failures and recalls.
- Configured workflows, environment variables, and subscriber details for email notifications.
- Backend and middleware fully functional; PDF attachments and alert data flow properly to subscribers.

◈ Machine Learning Model Evaluation

- Evaluated all ML models (Logistic Regression, Decision Tree, Random Forest).
- Finalized the **best performing model** (Random Forest) for device failure prediction.
- Integrated model outputs into the **dashboard**, showing risk metrics and probability distributions.
- Updated documentation with model evaluation results and finalization details.

◈ Tech Stack & Usage

Technology	Usage	Reason / Benefit
React.js	Frontend UI	Final dashboard, reports, visualization of ML predictions
Flask	Backend API layer	Connect dashboard to Snowflake; serve ML predictions and alert data
Snowflake	Cloud database	Real-time device, event, and manufacturer data storage
Novu	Notification platform	Email alerts for failures and recalls

Technology	Usage	Reason / Benefit
Random Forest Model	ML model	Predict device failure risk and generate dashboard insights
Python	Backend scripting & ML	Handle API logic, Snowflake connection, and ML execution
GitHub	Version control	Upload final dashboard, code, and documentation

◈ Day 10 - Teammate Contributions

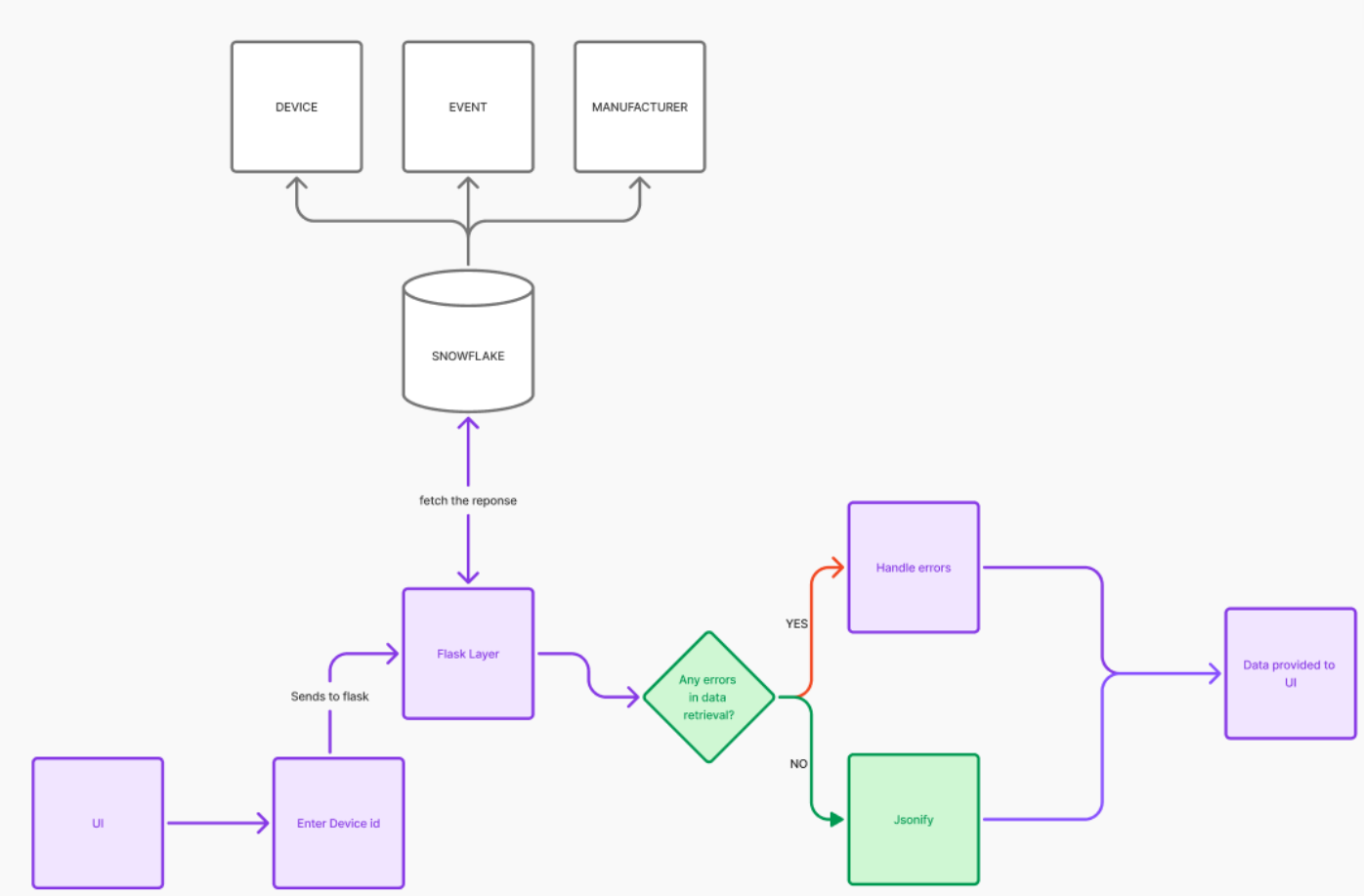
- **Teammate 1 -** Completed multi-page report module with timeline and tables.
- **Teammate 2 -** Finalized and uploaded Manufacturer Dashboard with full features.
- **Teammate 3 –** Firebase integration completed.
- **Teammate 4 -** Integrated Manufacturer Dashboard fully with Flask backend and Snowflake; ensured all modules (overview, devices, analytics, reports) work seamlessly with live data; tested error handling, loading states, and responsiveness and the model integration has been completed.
- **Teammate 5 -** .Completed alert system with email notifications and workflow verification.
- **Teammate 6 –** Completed all the works.

◈ Day 10 – Summary

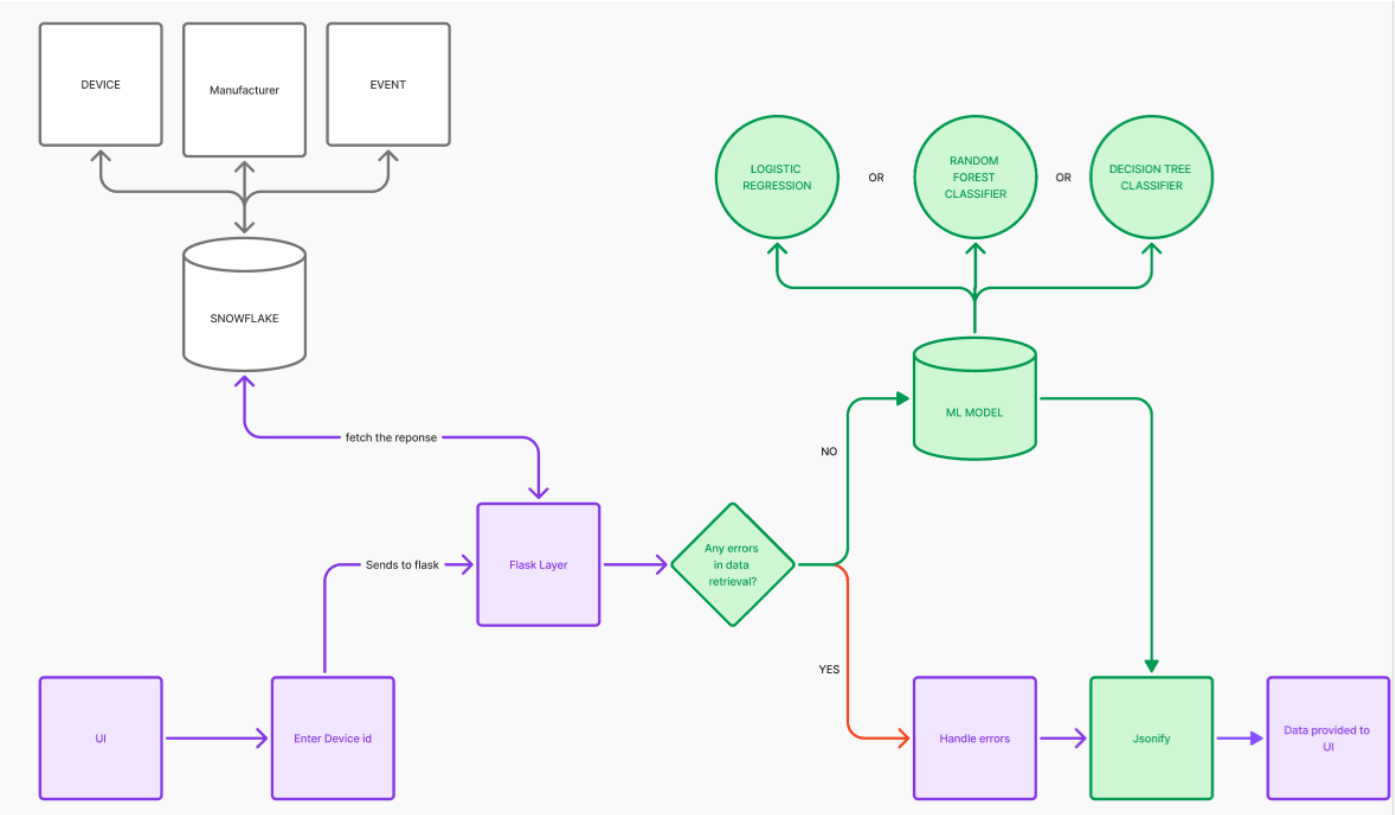
- Multi-page report module finalized and tested.
- Manufacturer Dashboard fully functional and uploaded to GitHub.
- Flask + Snowflake integration verified for live data display.
- Alerts system fully completed and tested via Novu.
- Random Forest model finalized and integrated for predictive analytics.
- Documentation updated; project ready for submission.

OVERALL WORKFLOW OF THE ENTIRE PROJECT

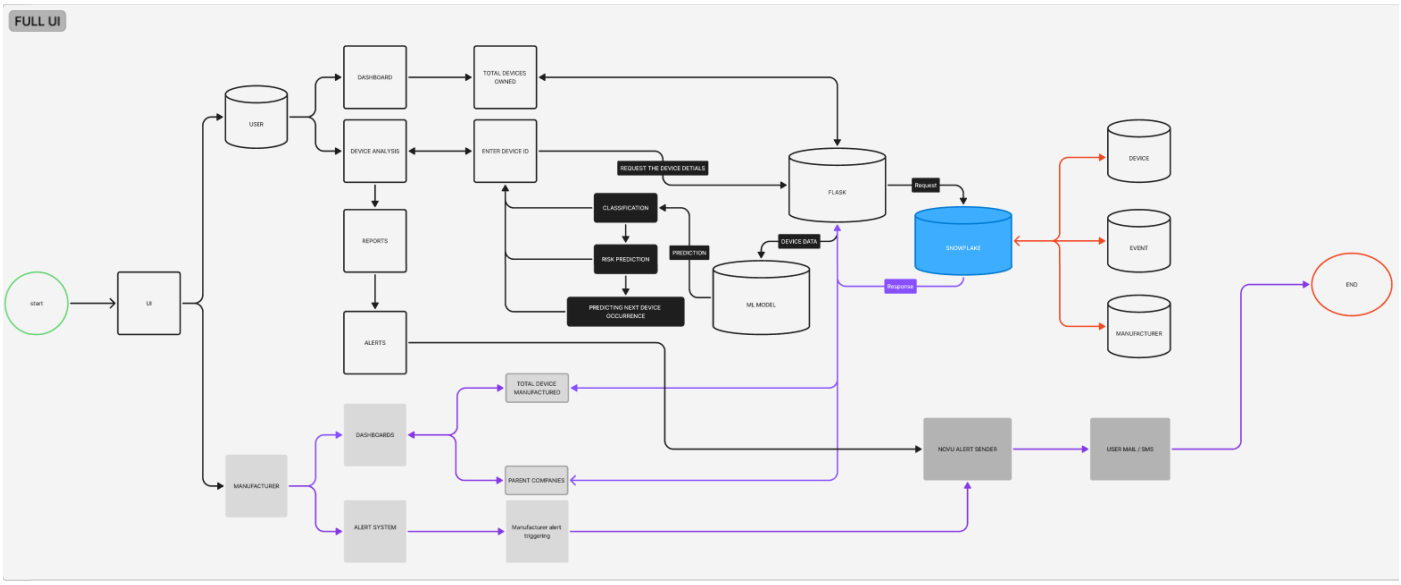
UI + FLASK



UI + FLASK + MODEL



ENTIRE FLOW:


FULL UI
start
UI
USER
DASHBOARD
DEVICE ANALYSIS
REPORTS
ALERTS
TOTAL DEVICES OWNED
ENTER DEVICE ID
REQUEST THE DEVICE DETIALS
CLASSIFICATION
RISK PREDICTION
PREDICTING NEXT DEVICE OCCURRENCE
PREDICTION
ML MODEL
DEVICE DATA
FLASK
Request
Response
SNOWFLAKE
DEVICE
EVENT
MANUFACTURER
END
MANUFACTURER
DASHBOARDS
ALERT SYSTEM
TOTAL DEVICE MANUFACTURED
PARENT COMPANIES
Manufacturer alert triggering
NOVU ALERT SENDER
USER MAIL / SMS

DAY 11 – HACKATHON PROGRESS UPDATE

The mentor reviewed the overall individual progress during the meeting and suggested areas for improvement in the project. The overall integration work was also examined and discussed. Additionally, the mentor provided ideas and raised questions to help us clarify and strengthen our situation analysis.

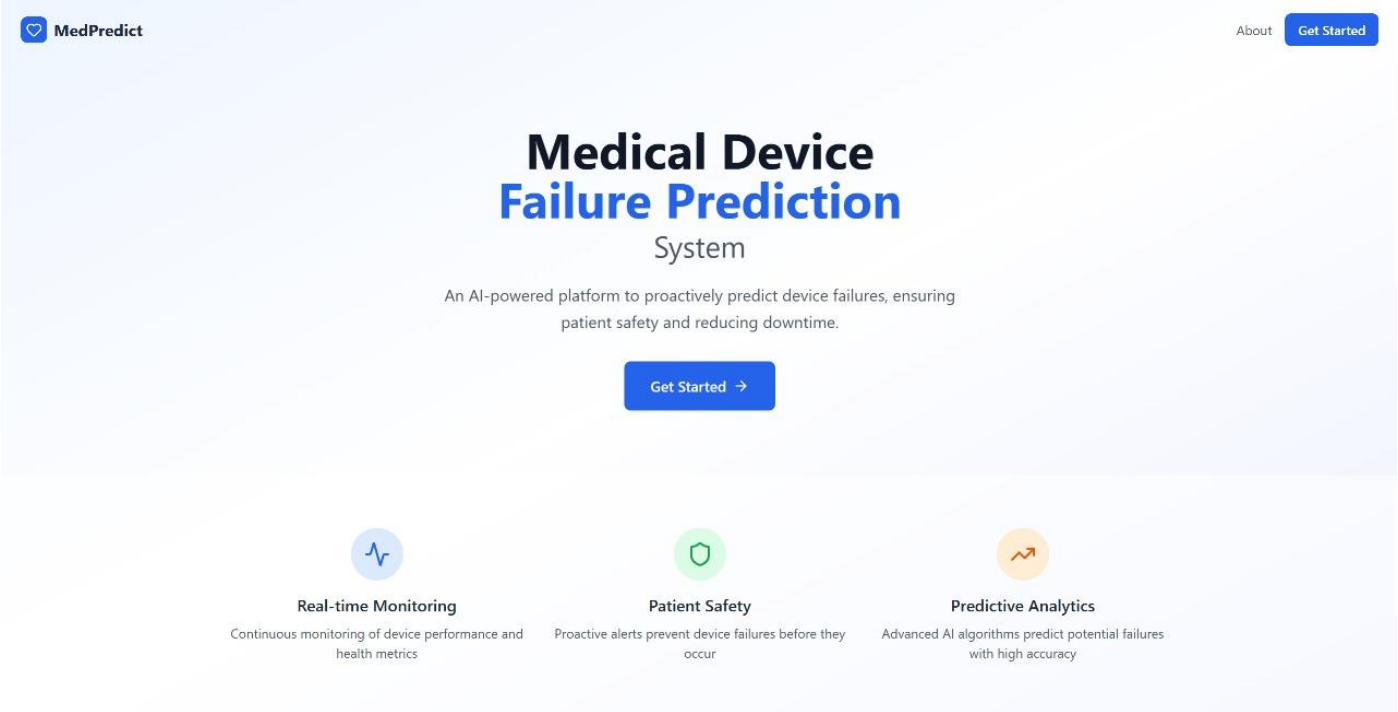
Progress:

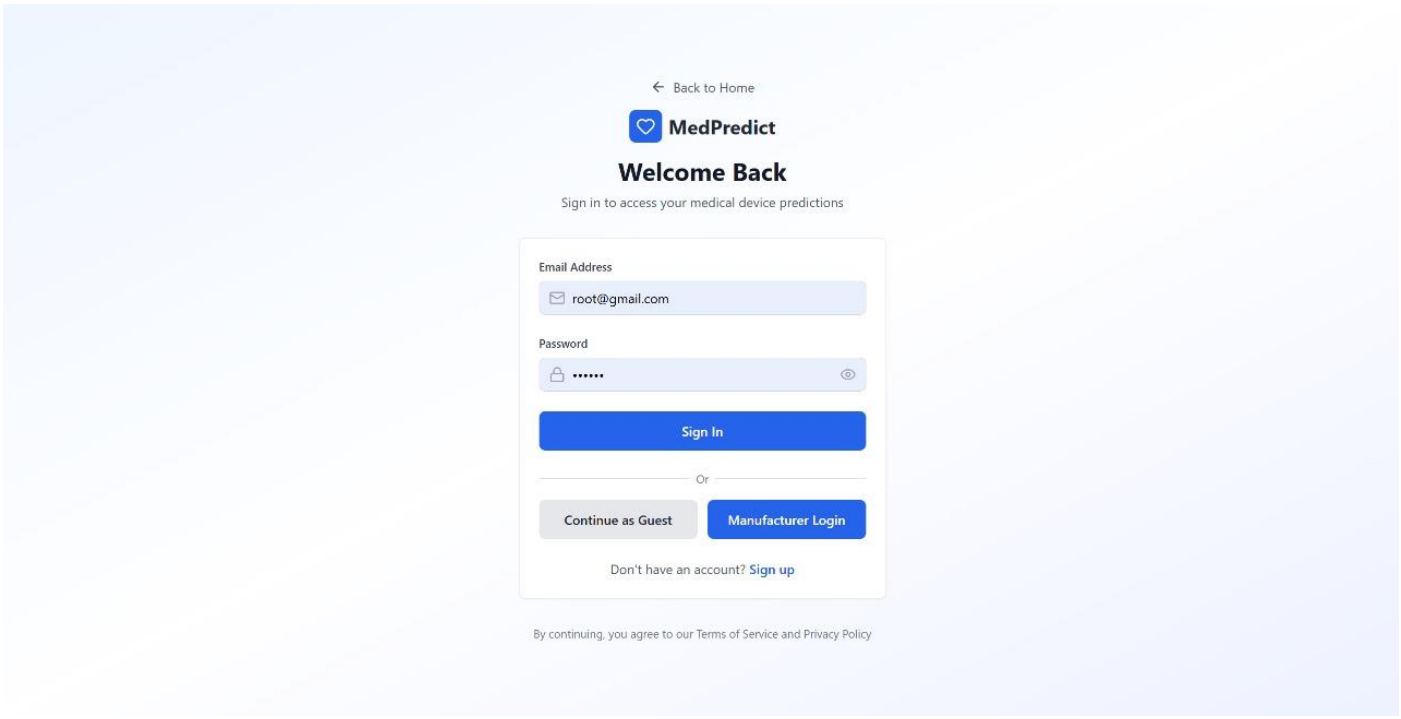
The AWS component has been implemented, and the report has been uploaded to the S3 bucket for future reference

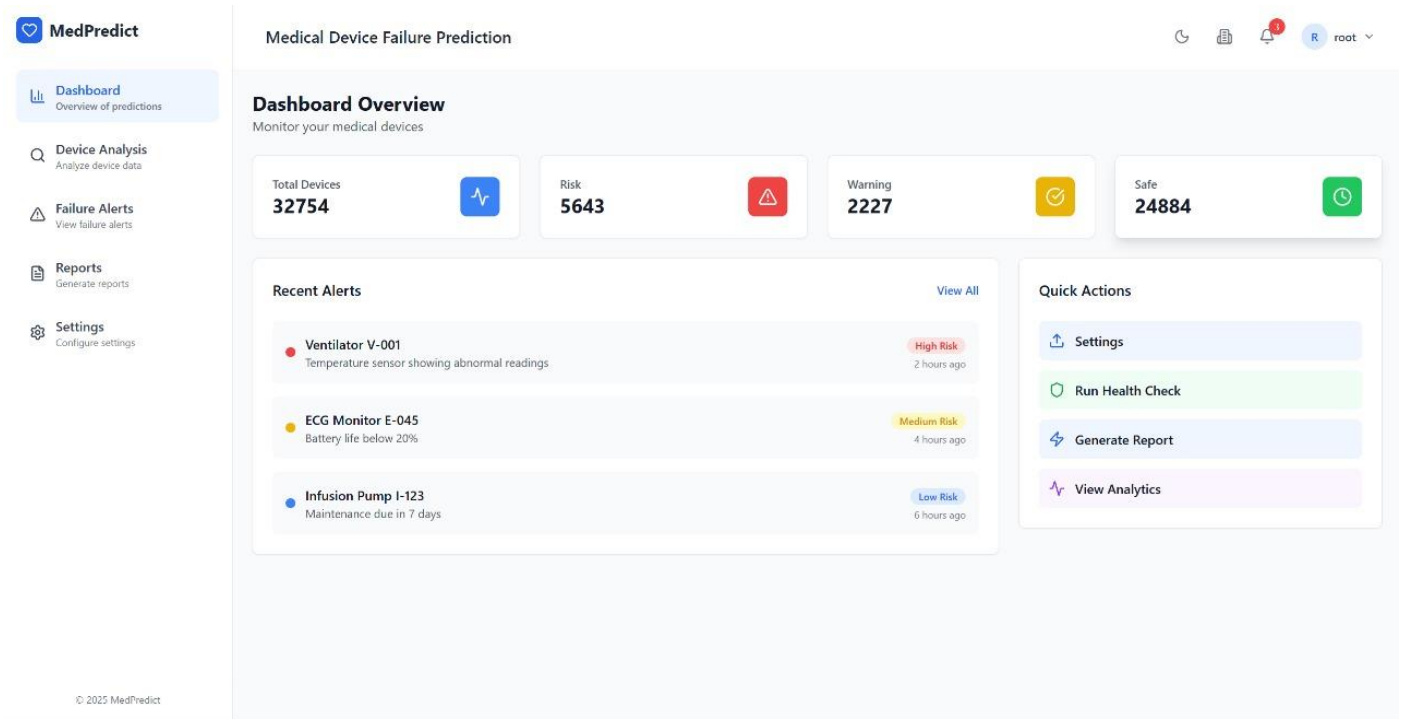
DAY 12 – HACKATHON PROGRESS UPDATE

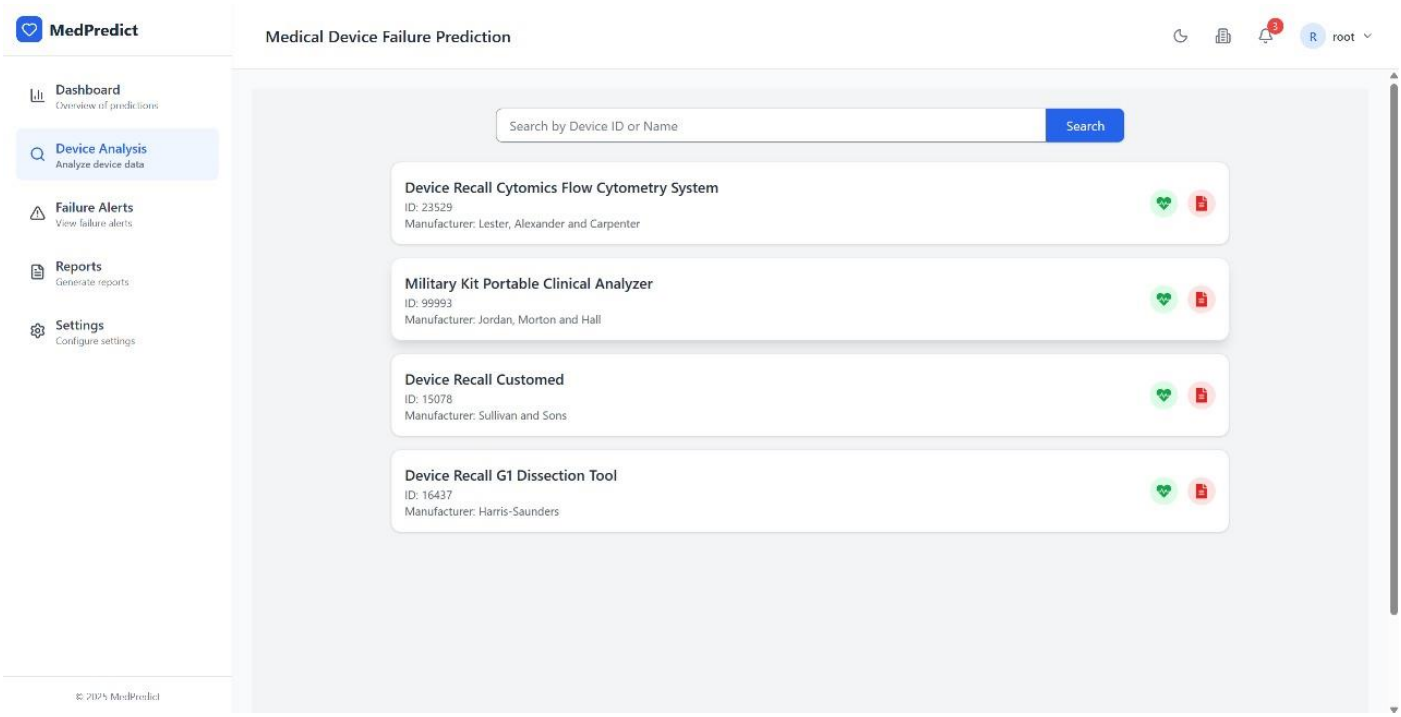
The improvements have been made based on the mentor's suggestions from Day 11. We applied the mentor's feedback and finalized the submissions.

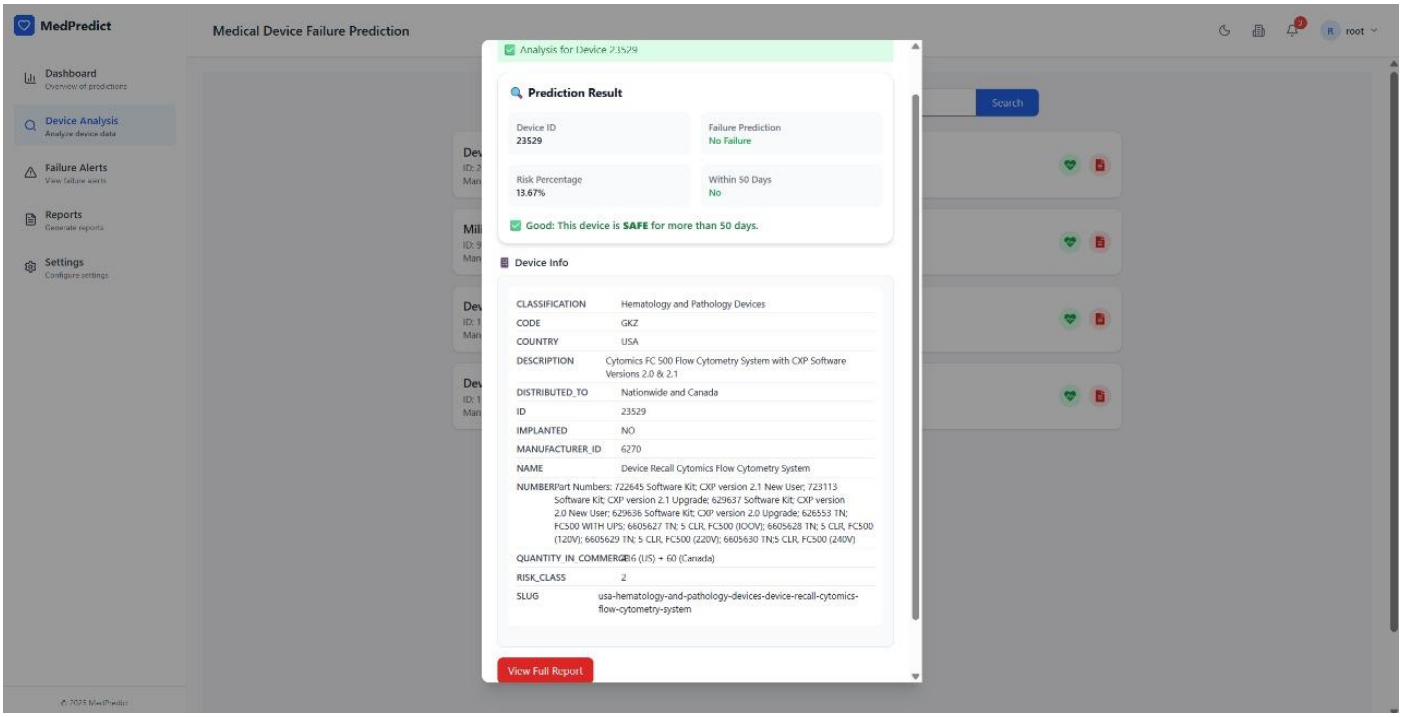
REACT UI




Back to Home
MedPredict
Welcome Back
Sign in to access your medical device predictions
Email Address
root@gmail.com
Password
Sign In
Or
Continue as Guest
Manufacturer Login
Don't have an account? Sign up
By continuing, you agree to our Terms of Service and Privacy Policy


MedPredict
Medical Device Failure Prediction
root
Dashboard
Overview of predictions
Device Analysis
Analyze device data
Failure Alerts
View failure alerts
Reports
Generate reports
Settings
Configure settings
Dashboard Overview
Monitor your medical devices
Total Devices
32754
Risk
5643
Warning
2227
Safe
24884
Recent Alerts
View All
Ventilator V-001
Temperature sensor showing abnormal readings
High Risk
2 hours ago
ECG Monitor E-045
Battery life below 20%
Medium Risk
4 hours ago
Infusion Pump I-123
Maintenance due in 7 days
Low Risk
6 hours ago
Quick Actions
Settings
Run Health Check
Generate Report
View Analytics
© 2025 MedPredict


MedPredict
Medical Device Failure Prediction
root
Dashboard
Overview of predictions
Device Analysis
Analyze device data
Failure Alerts
View failure alerts
Reports
Generate reports
Settings
Configure settings
Search by Device ID or Name
Search
Device Recall Cytomics Flow Cytometry System
ID: 23529
Manufacturer: Lester, Alexander and Carpenter
Military Kit Portable Clinical Analyzer
ID: 99993
Manufacturer: Jordan, Morton and Hall
Device Recall Customed
ID: 15078
Manufacturer: Sullivan and Sons
Device Recall G1 Dissection Tool
ID: 16437
Manufacturer: Harris-Saunders


MedPredict
Medical Device Failure Prediction
root
Analysis for Device 23529
Prediction Result
Device ID
23529
Failure Prediction
No Failure
Risk Percentage
13.67%
Within 50 Days
No
Good: This device is SAFE for more than 50 days.
Device Info
CLASSIFICATION
Hematology and Pathology Devices
CODE
GKZ
COUNTRY
USA
DESCRIPTION
Cytomics FC 500 Flow Cytometry System with CXP Software Versions 2.0 & 2.1
DISTRIBUTED_TO
Nationwide and Canada
ID
23529
IMPLANTED
NO
MANUFACTURER_ID
6270
NAME
Device Recall Cytomics Flow Cytometry System
NUMBER
Part Numbers: 722645 Software Kit; CXP version 2.1 New User; 723113 Software Kit; CXP version 2.1 Upgrade; 629637 Software Kit; CXP version 2.0 New User; 629636 Software Kit; CXP version 2.0 Upgrade; 626553 TN; FC500 WITH UPS; 6605627 TN; 5 CLR, FC500 (IOOV); 6605628 TN; 5 CLR, FC500 (120V); 6605629 TN; 5 CLR, FC500 (220V); 6605630 TN;5 CLR, FC500 (240V)
QUANTITY_IN_COMMERCE
86 (US) + 60 (Canada)
RISK_CLASS
2
SLUG
usa-hematology-and-pathology-devices-device-recall-cytomics-flow-cytometry-system
View Full Report

Device Analysis Report

Device ID: **23529** Generated: 2025-09-02 23:56:13 Source: Snowflake

Prediction Health

Low Risk

Risk %: **13.67**
Within 50 days: **No**

Manufacturer

No manufacturer data available

Device Information

CLASSIFICATION : Hematology and Pathology Devices

CODE : GKZ

COUNTRY : USA

DESCRIPTION : Cytomics FC 500 Flow Cytometry System with CXP Software Versions 2.0 & 2.1

DISTRIBUTED_TO : Nationwide and Canada

ID : 23529

IMPLANTED : NO

MANUFACTURER_ID : 6270

NAME : Device Recall Cytomics Flow Cytometry System

NUMBER : Part Numbers: 722645 Software Kit; CXP version 2.1 New User; 723113 Software Kit; CXP version 2.1 Upgrade; 629637 Software Kit; CXP version 2.0 New User; 629636 Software Kit; CXP version 2.0 Upgrade; 626553 TN; FC500 WITH UPS; 6605627 TN; 5 CLR, FC500 (IOOV); 6605628 TN; 5 CLR, FC500 (120V); 6605629 TN; 5 CLR, FC500 (220V); 6605630 TN;5 CLR, FC500 (240V)

QUANTITY_IN_COMMERCE : 416 (US) + 60 (Canada)

RISK_CLASS : 2

SLUG : usa-hematology-and-pathology-devices-device-recall-cytomics-flow-cytometry-system

Related Events

9/2/25, 11:56 PM Device Report - 23529

Device Analysis Report

Device ID: 23529 Generated: 2025-09-02 23:56:13 Source: Snowflake

Prediction Health

Low Risk

Risk %: 13.67
Within 50 days: No

Manufacturer

No manufacturer data available

Device Information

CLASSIFICATION : Hematology and Pathology Devices

CODE : GKZ

COUNTRY : USA

DESCRIPTION : Cytomics FC 500 Flow Cytometry System with CXP Software Versions 2.0 & 2.1

DISTRIBUTED_TO : Nationwide and Canada

ID : 23529

IMPLANTED : NO

MANUFACTURER_ID : 6270

NAME : Device Recall Cytomics Flow Cytometry System

NUMBER : Part Numbers: 722645 Software Kit; CXP version 2.1 New User; 723113 Software Kit; CXP version 2.1 Upgrade; 629637 Software Kit; CXP version 2.0 New User; 629636 Software Kit; CXP version 2.0 Upgrade; 626553 TN; FC500 WITH UPS; 6605627 TN; 5 CLR, FC500 (IOOV); 6605628 TN; 5 CLR, FC500 (120V); 6605629 TN; 5 CLR, FC500 (220V); 6605630 TN;5 CLR, FC500 (240V)

QUANTITY_IN_COMMERCE : 416 (US) + 60 (Canada)

RISK_CLASS : 2

SLUG : usa-hematology-and-pathology-devices-device-recall-cytomics-flow-cytometry-system

1

Print 2 sheets of paper

Destination Microsoft Print to PDF

Pages All

Layout Portrait

Color Color

More settings

Print Cancel